**Loading configuration files**

Hibernate configuration file (hibernate.cfg.xml) should have entry of mapping file (Student.hbm.xml) and obviously hbm file should have mapping of table and model-class.

To load hibernate.cfg.xml **configure** method of **Configuration** class is used.

sessionFactory = **new** Configuration().configure("/*Hibernate*/A\_Basics/

hibernate.cfg.A\_Basics.xml").buildSessionFactory();

This returns sessionFactory object. See the builder pattern.

**Loading configuration files**

Configuring the location/name of hibernate configuration file.

Lets say location of files are as following in project dir structure:

* model class - src/Hibernate/A\_Basics/Student
* cfg xml file - src/Hibernate/hibernate.cfg.A\_Basics.xml
  + by default it is - src/hibernate.cfg.xml
* mapping file - src/Hibernate/Student.hbm.xml
  + by default it is - src/Student.hbm.xml

Then to ensure all the above is loaded correctly, do the following :

* In Student.hbm.xml provide fully qualified model class name.

<hibernate-mapping>

<class name=*"Hibernate.A\_Basics.Student"* table=*"STUDENT"*>

.......

</class>

</hibernate-mapping>

* In hibernate.cfg.A\_Basics.xml provide full path of mapping file

<mapping resource=*"Hibernate/Student.hbm.xml"*/>

* To load hibernate.cfg.A\_Basics.xml

sessionFactory = **new** Configuration().configure("/*Hibernate*/A\_Basics/

hibernate.cfg.A\_Basics.xml").buildSessionFactory();

In default-case step 2 and 3 is not required.

If any of the above value is wrong we get following **MappingException**.

Exception in thread "main" **org.hibernate.MappingException**: Unknown entity: A\_Basics.Student

at org.hibernate.impl.SessionFactoryImpl.getEntityPersister (SessionFactoryImpl.java:693)

at org.hibernate.impl.SessionImpl.getEntityPersister(SessionImpl.java:1485)

.

.

at org.hibernate.impl.SessionImpl.save(SessionImpl.java:701)

at org.hibernate.impl.SessionImpl.save(SessionImpl.java:697)

at A\_Basics.Mainz.main(Mainz.java:29)

**Annotation**:

**By default**

* class name is table name in database
* Variable/property name is column name in database.
* column is nullable except primary key
* Java data type Date is mapped with SQL data type Date with time-stamp.

Annotations at class level

* @Etity - class written to persist or whole data into or from the database.
* @Table = if database table name is other than class name then this can be mapped in table annotation.
  + @Table(name = "STUDENT\_ANNOTATION")

Annotations at property level

* @Id - Applied to property. specify the primary key.
* @Column - @Column (name = "FIRST\_NAME", nullable=**false**)
  + name - if database column name is other than class variable name.
  + nullable - if set false, column will not be nullable.
* @Transient - property will not be considered for database table-column.
* @Temporal - Using TemporalType, java data type Date is mapped with SQL data type Date with differnt Formats.
  + @Temporal(TemporalType.***DATE***)

hbm2ddl.auto : It has 4 values

* **Create** – Every time new table will be created using model table object. Previous data would lost.
  + Tecnicaly hibernate will drop off existing schema of database(along with tables) if present, at the time of creating **SessionFactory** object when you run the program
  + If database has schema or table which has no entry as model object, those will not be dropped-off.
* CreateDrop : Does the exactly same as Create.
  + When **SessionFactory** object is explicitly closed then hibernate would drop off schema and all data.
* Update : updates the schema with new value/row etc.
  + for experimental purpose as of now, don't use in production.
* Validate :
  + hbm2ddl property is not recommended in production. Don't use this property in hibernate.cfg.xml file.

So when you are running first time hibernate program(such that tables are not present) set hbm2ddl property as create and next time onwards modify hbm2ddl property as update.

If you do this you need not to create the tables manually in DB.

**Relationships**

One to one mapping: Table A and B is said to have one to one mapping iff:

* Each row of table A is linked to another row of table B.
* No. of row of table A is equal to no. of row of table B.

Ex: Each row of student is linked with their details.

* They are linked with student\_id.
* Student\_id in student detail table is foreign key.

Establishing one to one through hibernate.

* Declare Student(Parent) class into StudentDetails(Child) along with it's setters and getters.
* Mark OneToOne declaration at the top of this annotation.
* Set Student to StudentDetails before persisting.

Prerequisite in database:

* Create the tables Student and StudetDetails.
  + CREATE TABLE Student (name VARCHAR(20), student\_id int(20));
  + CREATE TABLE StudentDetails (student\_id int(20), address VARCHAR(20), mobile VARCHAR(20), dob VARCHAR(20));
  + Alter TABLE Student Add Primary Key (student\_id)
  + Alter TABLE StudentDetails Add Primary Key (student\_id)
* Establish Foreign key reference in StudentDetails
  + Alter TABLE StudentDetails
  + add Foreign Key (student\_id)
  + References Student(student\_id)
* Make Primary-key for both the table auto-increment:
  + Alter TABLE Student modify column student\_id INT auto\_increment

Annotations:

OneToOne – on property(Parent class) with which one-to-one relationship has to be established.

JoinColumn – Provide the column name(property of Parent class) on which -to-one relationship has to be established.

@JoinColumn(name = "student\_id")

When we try to persist child and parent class into DB(through one-to-one relationship) hibernate must generate same id for both. How do we ensure this?

Using GenericGenerator

@GenericGenerator(name = "newGenerator", strategy = "foreign", parameters = { @Parameter(value = "student", name = "property")})

Why do we need to establish relationship(OnetoOne) at java code level as table level it is already defined using foreign-key reference?

Operation perform on the child at java code will also perform the all desired operation on parent class

If you would not have established this relationship(OnetoOne) at java code level then you might have to perform

CascadeType

Uni/Bi directional OneToOne Mapping.

**Hibernate object states**

An (Hibernate) object can have three states: transient, persistent and detached.

* **Transient** - When Object is just created (i.e. not associated with session) and doesn’t have corresponding record in table.
  + When Object is just created and doesn’t have primary key, the state is called transient.
  + Object creation till before session.save(Object).
  + Object is associated with session using - session.save(Obj).
  + In transient state while saving the object, insert query is fired.
* **Persistent** – Corresponding record is available in table and object is associated with session.
  + When the session is opened and the object is just saved in or retrieved from the database.
  + From session.save() till session.close()
  + In persistent state update query is fired on already inserted record (because of transient state insertion) to synch DB record.
* Detached – Corresponding record is available in table and object is not associated with session.
  + When the session is closed, the state changes to detached.
  + After session.close().
  + In detached state there would not be any affect of query on DB.

If in transient record is not inserted (because record is not set) then while saving we get exception.

When you get an object using session.get() then that object leaves in persistent object till session.close(). Where state changes to detached.

In persistent state if you make any change(Note: session.update(student) is not executed but just change is made through setters) in the record, update query is fired on corresponding record in db-table to synch that record into table.

Converting object's state

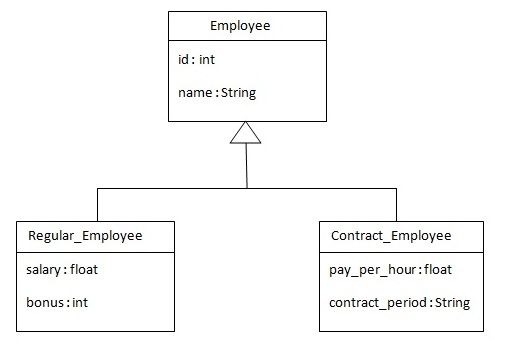
* Persistent to transient
  + When you delete the object using session.delete() then the object's state changes back to transient.
* Detached to persistent: you need to re attach the object from new session
  + By creating new session and
  + Passing the object to the **update** method of that session.

Hibernate Inheritance Mapping

There are three inheritance mapping strategies to map the inheritance hierarchy classes with the table of the database:

* Table Per Hierarchy : **single table** is required to map the whole hierarchy,
  + an extra column (discriminator column) is added to identify the class.
  + Nullable values are stored in the table
  + subelement of class - **subclass**
* Table Per Concrete class : tables are created as per class.
  + Duplicate column is added in subclass tables.
  + subelement of class - **union-subclass**
* Table Per Subclass : tables are created as per class but related by foreign key.
  + So there are no duplicate columns.
  + subelement of class - **subclass**

**Table Per Hierarchy:** whole hierarchy of the parent-child class is mapped into one table of the database.



Below is the mapping file for this hierarchy

<hibernate-mapping>

<**class** name="com.javatpoint.mypackage.Employee" table="emp121" discriminator-value="emp">

<id name="id">

<generator **class**="increment"></generator>

</id>

<discriminator column="type" type="string"></discriminator>

<property name="name"></property>

<**subclass** name="com.javatpoint.mypackage.Regular\_Employee" discriminator-value="reg\_emp">

<property name="salary"></property>

<property name="bonus"></property>

</subclass>

<**subclass** name="com.javatpoint.mypackage.Contract\_Employee" discriminator-value="con\_emp">

<property name="pay\_per\_hour"></property>

<property name="contract\_duration"></property>

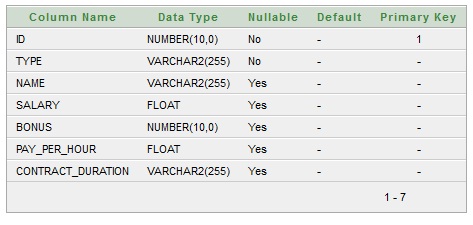
</subclass>

</**class**>

</hibernate-mapping>

* There are three classes in this hierarchy.
  + Employee is the super class.
  + Regular\_Employee and
  + Contract\_Employee classes.
* By this inheritance strategy, we can map the whole hierarchy by single table only.
  + Here, an extra column (discriminator column) is created in the table to identify the class.
  + To specify this, discriminator subelement of class must be specified.
* The **subclass** subelement of class, specifies the subclass.
  + In this case, Regular\_Employee and Contract\_Employee are the subclasses of Employee class.

**Table structure for above hierarchy:**



Output:



**Anotation:**

@Inheritance(strategy=InheritanceType.SINGLE\_TABLE)

@DiscriminatorColumn(name="type",discriminatorType=DiscriminatorType.STRING)

@DiscriminatorValue(value="employee")

**public** **class** Employee {   }

@DiscriminatorValue("regularemployee")

**public** **class** Regular\_Employee **extends** Employee{  }

**Table Per Concrete class:**

There will be three tables in the database, having no relations to each other, each representing a particular class.

The **union-subclass** subelement of class, specifies the subclass.

It adds the columns of parent table into this table. In other words, it is working as a union.

<hibernate-mapping>

<class name=*"X\_Inherit.B\_TB\_Concrete.XML.Employee"* table=*"Concr\_emp122"*>

<id name=*"id"*>

<generator class=*"increment"*></generator>

</id>

<property name=*"name"*></property>

<**union-subclass** name=*"X\_Inherit.B\_TB\_Concrete.XML.Regular\_Employee"*

table=*"Concr\_regemp122"*>

<property name=*"salary"*></property>

<property name=*"bonus"*></property>

</union-subclass>

<**union-subclass** name=*"X\_Inherit.B\_TB\_Concrete.XML.Contract\_Employee"*

table=*"Concr\_contemp122"*>

<property name=*"pay\_per\_hour"*></property>

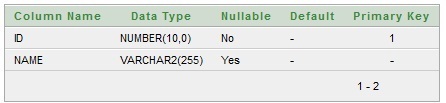
<property name=*"contract\_duration"*></property>

</union-subclass>

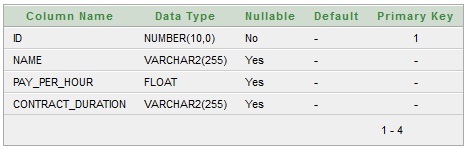
</class>

</hibernate-mapping>

Table structure for Employee class



#### Table structure for Regular\_Employee class



**Table Per Subclass**: subclass mapped tables are related to parent class mapped table by primary key and foreign key relationship.

The **<joined-subclass>** element of class is used to map the child class with parent using the primary key and foreign key relation.

TODO:

Check the behaviour, If following annotations are not used

**@Entity**

Model class configured in CFG.xml file is not identified. And MappingException for unknown entity is thrown.

Exception in thread "main" org.hibernate.MappingException: Unknown entity: C\_PKAutoGen.StudentAnnotation at org.hibernate.impl.SessionFactoryImpl.getEntityPersister(SessionFactoryImpl.java:693) at org.hibernate.impl.SessionImpl.getEntityPersister(SessionImpl.java:1485)

@GeneratedValue

Issues: